

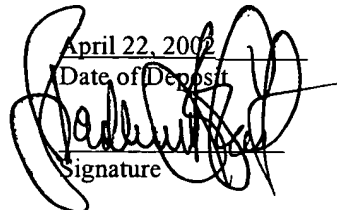
#4
7-3-02

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant : Amrhein et al.
Serial No. : 10/056,855 Examiner: t/b/a
Filed : November 13, 2001 Group Art Unit: 2182
For : APPARATUS AND METHOD FOR CHECKING
THE STATUS OF CONTROL SYSTEMS

SUBMISSION OF PRIORITY DOCUMENTS

I hereby certify that this paper is being deposited with the United States Postal Service as First Class Mail in an envelope addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231 on:

April 22, 2002
Date of Deposit

Signature

Bradley B. Geist
Attorney Name

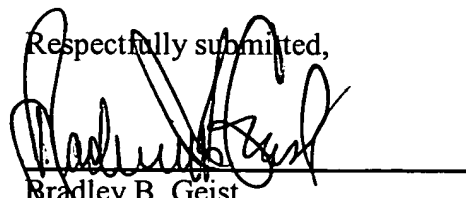
27,551
Registration No.

Assistant Commissioner for Patents
Washington, D.C. 20231

Sir:

We enclose herewith German patent application nos. 100 63 044.8 and 101 25 385.0 which are the priority documents for the above referenced patent application.

Respectfully submitted,


Bradley B. Geist
Patent Office Reg. No. 27,551
Attorneys for Applicants
(212) 408-2562



**Prioritätsbescheinigung über die Einreichung
einer Patentanmeldung**

Aktenzeichen: 100 63 044.8

Anmeldetag: 18. Dezember 2000

Anmelder/Inhaber: Siemens Aktiengesellschaft, München/DE

Bezeichnung: Statusprogramm

IPC: G 06 F 11/36

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 17. Oktober 2001
Deutsches Patent- und Markenamt
Der Präsident
Im Auftrag

Hof

Beschreibung

Statusprogramm

Verfahren zum Debuggen von Programmen für industrielle Steuerungen und
DEBUG-Mechanismus für Steuerungen

Es wird eine Funktionalität angeboten, in der der Anwender einen Bereich eines ST-Programmes (Programm, das in Structured Text geschrieben ist) markieren kann. Aktiviert er den DEBUG-Mechanismus, werden ihm in einem zweiten Fenster neben seiner ST-Quelle für den entsprechend zyklisch durchlaufenen Programmcode noch die Werte der im Programmcode verwendeten Variablen konsistent für den jeweiligen Durchlauf angezeigt. Damit kann er sehr gut bei zyklischer SPS-Funktionalität Diagnose und Programm-Debug realisieren. Für eine Realisierung des Mechanismus scheidet eine interpretative Lösung aus Performance-Gründen aus.

Für das ST-Programm wird speziell instrumentierter Zwischencode erzeugt. Dieser Zwischencode enthält zusätzliche Informationen mit deren Hilfe später das Run-Time-System auf eben verwendete Variablen Rückschlüsse ziehen kann.

Der Compiler kompiliert das ST-Programm vor dem Download in die Steuerung und in der Steuerung wird dieser Zwischencode noch mal in mikroprozessorspezifischen Code umgesetzt.

Diese Compiler-Funktionalität ist im Basis-Engineeringsystem realisiert. Das Basis-Engineeringsystem hat in seiner Datenhaltung eine Zuordnungsinformation, welche ST-Programmzeilen zu welchen Zwischencodeabschnitten gehören. Der Anwender kann durch grafische Markierungen eine Abschnittskennung von ST-Programmzeilen vornehmen, darauf aufsetzend ermittelt ein Debugger den dazugehörigen Zwischencodebereich.

Die Zuordnung kann z.B. über eine Zuordnungstabelle oder einen Verweis erfolgen.

Auf ES-Seite (ES steht für Engineering System) existiert somit ein ST-Programm mit einem kompilierten Zwischencode, wo ich eben über diese Zuordnungstabelle jeweils zuordnen kann, welcher ST-Programmbereich welchem Zwischencodebereich entspricht. Dieser Zwischencode, der instrumentiert ist, wird dann ins Run-Time-System geladen. Das Run-Time-System setzt den Zwischencode dann noch mal in Mikroprozessorcode um. Dann kann das Programm im Prinzip ablaufen, d.h. CPU kann in Run gehen. Der Mikroprozessorcode wird zyklisch durchlaufen. Wenn nun der Anwender im ST-Programm einen bestimmten Bereich markiert, dann wird ein Auftrag an das Run-Time-System generiert, welcher Zwischencodebereich in Zukunft beobachtet werden soll. Das Run-Time ermittelt die Mikroprozessorcodebereiche aus dieser Information und richtet sozusagen die Infrastruktur für eine Programmbeobachtung ein. Nachdem dieser Dienst eingerichtet wurde, kann er zu ab diesem Zeitpunkt vom Anwender potenziell verwendet werden. Wird die Programmbeobachtungsfunktion vom Anwender explizit im Editor angestoßen, wird von diesem Zeitpunkt an ein Beobachtungsmodul eingerichtet, das sich im Run-Time-System befindet. Wenn das Programm in diesen Code eintritt, werden die enthaltenen Variablenwerte in entsprechende Datenpuffer geschrieben, und zwar so lange, wie sich das Programm in diesem Codebereich befindet. Wird der Score des Codebereichs verlassen, bekommt das Engineeringsystem (ES) von diesem Beobachtungsmodul eine Notification. Das Engineeringsystem holt sich den Datenpuffer zurück. Ein Debug-Modul im ES ermittelt aus der Zwischencodenzuordnung und diesem Datenpuffer die entsprechende Zeileninfo im ST-Programm und die Variablen können, wie vom Anwender gewünscht, angezeigt werden. Ist die Anzeige erfolgt, wird ein neuer Beobachtungsauftrag für diese eingerichtete Infrastruktur abgesetzt, und das Spiel beginnt von vorne. Von vorne heißt, der Zyklus oder die Flusssteuerung erfolgt eigentlich durch den Editor. Das ist auch so sinnvoll, weil der Anwender nicht so schnell das Pro-

gramm am Bildschirm verfolgen kann wie der SPS-Zyklus läuft. Eine Beobachtung im SPS-Zyklus würde außerdem nur die Run-Time-CPU stark belasten, würde für den Anwender keinen Vorteil bringen.

Von existierenden Technologien unterscheidet sich der vorgeschlagene Mechanismus im Wesentlichen dadurch, dass auf anderen Steuerungen Hardware-Unterstützung notwendig war, d.h. man musste immer über irgendwelche Breakpoint-Mechanismen oder Interrupts von Prozessoren das Debugging abfahren oder aber eine spezielle Hardware-Unterstützung verwenden. Bei interpretativ arbeitenden Steuerungen wurde das Debugging bisher interpretativ gelöst, was natürlich sehr starke Laufzeiteinflüsse hatte.

Der große Vorteil der Erfindung ist, dass man hardwareunabhängig arbeiten kann und dass der Mechanismus die CPU nur genauso stark belastet, wie der Programmeditor eben diese Beobachtung überhaupt ausnutzen kann. (Laufzeit kostet nur das Beobachten des Markierten Bereiches. Genau betrachtet wird auch nur für den markierten Bereich der Debugcode durchlaufen. Alle anderen Bereiche werden wie ohne Debugoption abgearbeitet)

Für den Anwender eignet sich diese Funktionalität und dieser Mechanismus sehr gut zum Debuggen zyklisch laufender Programme. Aber auch Motion Tasks, die nicht zyklisch laufen, können sehr gut beobachtet werden. Insbesondere kann durch einen zusätzlichen Mechanismus der Trace Buffer schon vor Verlassen des Scopes abgeholt werden. Man kann dadurch z.B. ermitteln, wann man mit welchen Parametern in synchrone Aufrufe gerät. Auch zum Finden von Deadlocks in Motion Tasks eignet sich das Verfahren und der dazugehörige Mechanismus.

Patentansprüche

1. Verfahren zum Debuggen, zur Beobachtung und zur Diagnose von Programmen zum Betrieb industrieller Steuerungen mit Engineering System und Runtime-System, gekennzeichnet, durch eine Untermenge folgender Merkmale:

- Verwendung von Structured Text
- Instrumentierung des Structured Text Codes
- Erzeugung eines Zwischencodes
- Markierung des relevanten Zwischencodebereiches
- Abbildung des Zwischencodebereiches in einen Microprozessor-codebereiche
- Verwendung eines Editors
- Verwendung eines Compilers

2. System zum Debuggen, zur Beobachtung und zur Diagnose von Programmen zum Betrieb industrieller Steuerungen mit Engineering System und Runtime-System, gekennzeichnet, durch eine Untermenge folgender Merkmale:

- Verwendung von Structured Text
- Instrumentierung des Structured Text Codes
- Erzeugung eines Zwischencodes
- Markierung des relevanten Zwischencodebereiches
- Abbildung des Zwischencodebereiches in einen Microprozessor-codebereiche
- Verwendung eines Editors
- Verwendung eines Compilers

Bilder Elm "Status Programm" 1/2

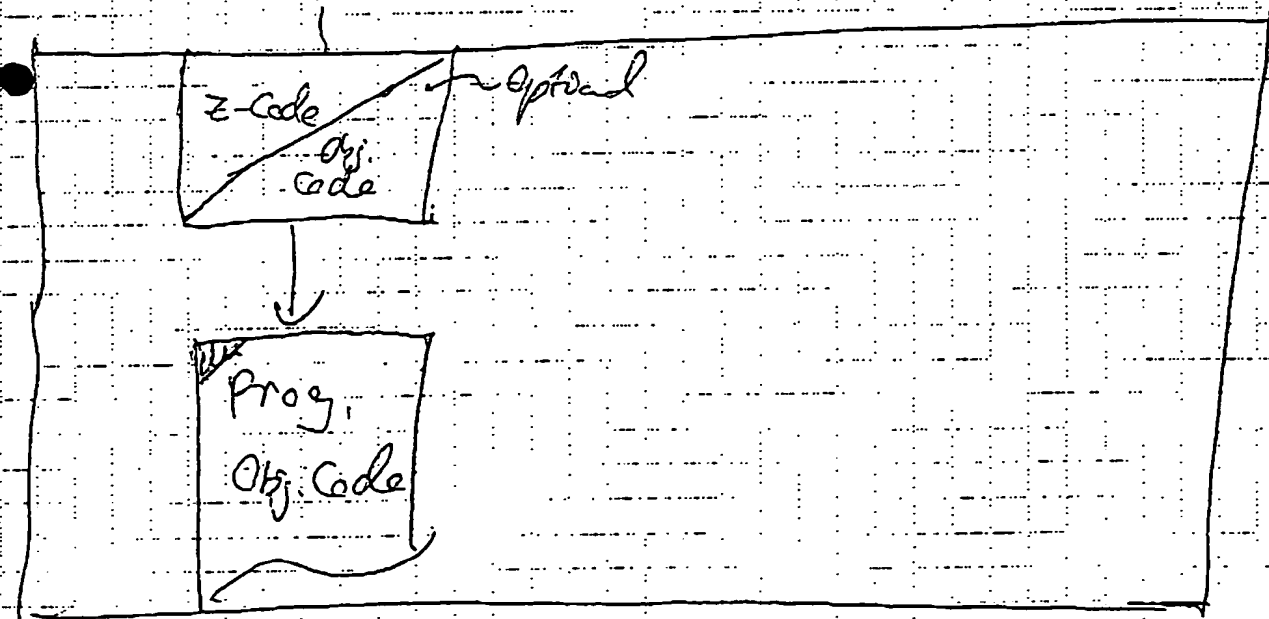
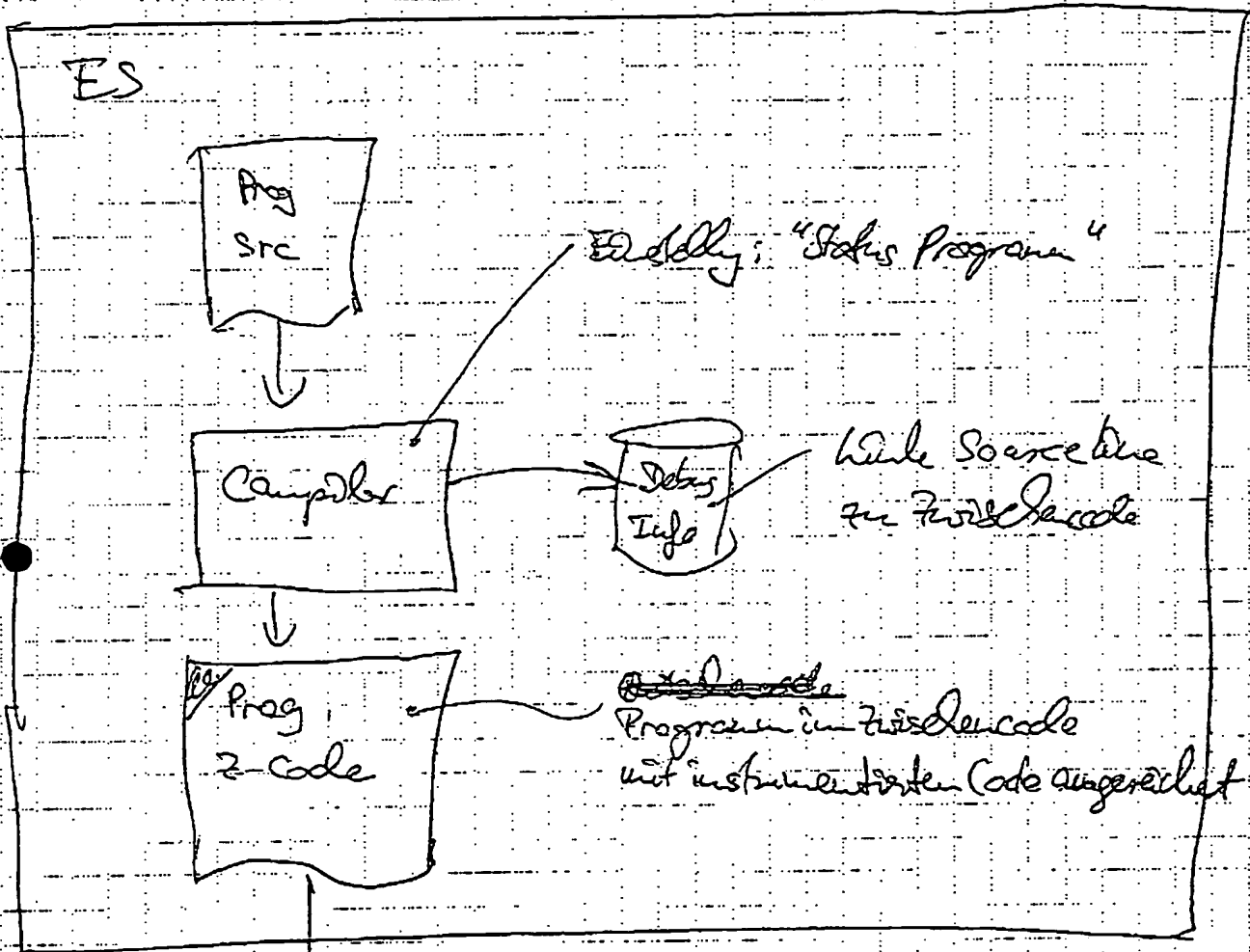


Bild 1: Anwendung des Quellprogramms um Debuginfo durch instrumentierten Code - 1 -

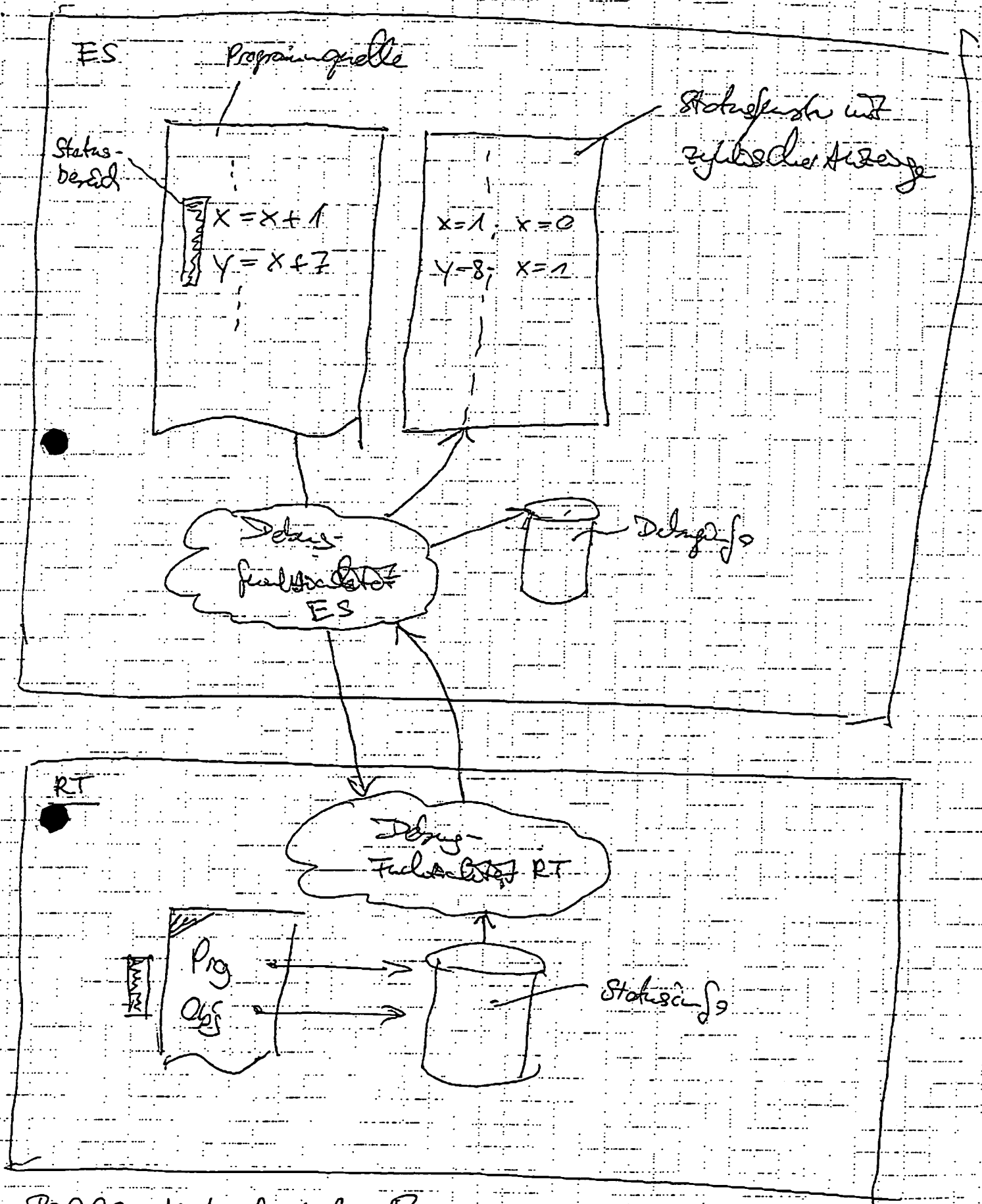


Bild 2: Ablauf statische Programm